presented at the 14th Workshop of the UK Planning and Scheduling Special Interest Group

CBR in Scheduling: Reusing Solution Components

Pádraig Cunningham
Department of Computer Science
Trinity College Dublin
College Green Dublin 2
Ireland
Padraig.Cunningham@cs.tcd.ie

Barry Smyth
Hitachi Dublin Laboratory
Trinity College Dublin
College Green Dublin 2
Ireland
Barry.Smyth@hdl.ie

Abstract

In this paper we explore the reuse of components of known good schedules in new scheduling problems. This involves accumulating a case-base of good quality schedules, retrieving a case (or cases) similar to a new scheduling problem and building a new schedule from components of the retrieved cases. Two CBR solutions to a single machine scheduling problem with schedule dependent setup times are described. These are evaluated by comparing them with two more conventional alternative techniques – simulated annealing and myopic search. Both CBR techniques are shown to provide good quality solutions and significant time improvements over simulated annealing.

1 Introduction

There is a lot of optimism at the moment about the usefulness of case-based reasoning (CBR) in the development of knowledge based systems. The view is that cases can represent good quality solutions that may be reused in new situations. Typically, cases represent compiled knowledge in weak theory domains (CAPLAN/CBC as described in (Muñoz, Paulokat & Wess, 1994) is a typical example of this idea.). There are other situations where solution reuse may be advantageous; for instance in optimisation problems where cases can represent highly optimised structures in the problem space. In this paper we explore this type of reuse in job scheduling problems.

Two distinct approaches to case-based scheduling can be identified from the literature. The first one is used in Koton's SMARTplan (1989). Cases are used to propose preliminary schedules which are then adapted (refined and repaired) to satisfy the target schedule requirements. The second approach does not uses cases to propose schedules, but instead to adapt schedules proposed by other methods (for example, Sycara & Miyashita, 1994). In other words cases can encode actual schedules (the first approach) or they can encode repair procedures (the second approach).

In this work we examine the first approach and look at two different modes of reuse -- (1) the reuse of single cases as *skeletal* solutions, and (2) the reuse of multiple cases as the *building blocks* of the desired target solution.

We are interested in the use of CBR in scheduling problems in general but we start here with a single machine problem where job setup time is schedule dependent. We consider two CBR solutions to this problem and we compare these with two standard solution techniques. The first is Simulated Annealing; this produces good quality solutions but is computationally expensive. The second is a naïve search mechanism that selects the nearest job next. This is referred to as the Myopic solution; it produces poorer solutions but is fast.

An extensive evaluation comparing these techniques is presented. Our conclusion is that the CBR techniques produce fairly good quality solutions quickly. The details of the scheduling problem are presented in the next section before the CBR solutions are introduced in section

3. The case retrieval mechanisms are described is section 4 and the evaluation is presented in section 5.

2 The Scheduling Problem

The scheduling problem addressed here involves scheduling m jobs on a single machine in a situation where job setup time is sequence dependent. The manifestation of this problem with which we are concerned arises in electronics assembly where the job setup time on component sequencing machines used in PCB assembly is schedule dependent. The setup time associated with scheduling job B after job A depends directly on the number of components required in B that were not used in A (see Cunningham & Browne, 1987 for more details). This is evidently an asymmetric Travelling Salesman Problem involving open 'tours'.

The data used in the experiments detailed in this paper describe a hypothetical situation. There is a total of 500 jobs that may need to be scheduled from time to time. Each job has between 40 and 80 components taken from a set of 200 components. The cost of scheduling job **B** after job **A** is represented by the number of components in **B** not in **A**, as said before. There is also a case-base of 600 good quality schedules of 100 jobs each taken from the population of 500 jobs. These schedules were produced using simulated annealing (see Cunningham & Smyth, 1995 for details of the algorithm).

3 Two Alternative CBR Solutions

In this paper we consider two alternative CBR solutions to this scheduling problem. The first solution we call the skeletal solution. This reflects the way in which the best matching case retrieved from the case-base is used to form a skeleton on which the new schedule is built. In this situation only one case from the case base is retrieved; the jobs that this case has in common with the target case make up the skeleton and the remaining cases are added to this skeleton using the following algorithm:-

```
function Add-jobs(schedule, rem-jobs)
{
    if rem-jobs {
          best-job ← Select-best-job(schedule, rem-jobs)
          rem-jobs ← Remove-job(best-job, rem-jobs)
          schedule ← Insert-job(schedule, best-job)
          Add-jobs(schedule, rem-jobs)
    }
}
```

The `Select-best-job` function takes each remaining job in turn and finds the point in the schedule where it can be inserted with the minimum cost. The best job is the one that can be inserted in the schedule with least cost.

The other CBR approach is radically different in that several cases are used in building a solution to the target problem. We call this the building block solution because sub-sections of the target schedule are discovered in cases in the case-base and composed into a new schedule to satisfy the target problem. The first step in this process is to retrieve these sub-sections of schedule from the case-base (see section 4 for more details). These schedule bits are contiguous sections of existing optimised schedules that contain only jobs on the target schedule.

```
  schedule ← starting job
  function Add-bits(schedule, bits-pool)
  {
      if bits-pool{
            best-bit ← Select-best-bit(schedule-end, bits-pool)
            if Close-to-schedule(best-bit){
                  schedule ← Concatenate(schedule, best-bit)
                  bits-pool ← Remove-unusable-bits(schedule, bits-pool)
                  Add-bits(schedule, bits-pool)
                  }
      }
}
```

The `Select-best-bit` function tries two things; first it looks for the longest bit that starts at the current end point of the incomplete schedule (same job), if it fails on this is selects the bit that starts closest to the current end point. Each time the schedule is extended, bits are removed from the pool if they contain jobs that are now already in the schedule. This process produces a schedule that is still missing some jobs; these jobs are added in using the process described in the skeletal solution.

We feel this building block approach is the more important for scheduling in general because it involves reusing fragments of optimised structure in new situations. This strategy has the potential for reuse in more complex scheduling problems.

4 Case Retrieval

Two different approaches to retrieval are described. One works to select a single best skeletal case which is reused as a template for the target solution; some jobs will be dropped from the template, some will be added. The other approach returns a collection of solution sequences, each of which may be used intact to produce a part of the target schedule.

4.1 Case Retrieval for the Skeletal Solution

Selecting a skeletal solution involves searching for a case which satisfies two criteria. First, the case must share a significant portion of the target jobs. Second, these jobs must be distributed as evenly as possible throughout the case solution – concentrations of target jobs do not make good skeletal solutions.

The pseudo-code below describes how the case-base is searched for the best possible skeleton as a two step process. The first step, *base filtering*, only accepts cases whose overlap with the target case suggests the possibility of a good skeletal solution; if a case intersects with the target in only a few jobs then it is unlikely to yield a suitable skeletal solution.

```
  function Retrieve-Skeleton(target-jobs, case-base) {
    base-cases ← Base-Filter(target-jobs, case-base)
    best-case ← First(base-cases)
    best-score ←  Skeletal-Quality(best-case)
    For each base-case in Rest(base-cases) {
      base-score ←  Skeletal-Quality(target-jobs, base-case)
      if base-score > best-score {
        best-score ←  base-score
        best-case ←  base-case
      }
    }
```

The second step chooses a single case by using a metric that measures skeletal solution quality in terms of the number of jobs in the skeletal solution and the evenness of the distribution of these jobs through the entire base solution.

```
function Skeletal-Quality(target-jobs, base-case) {
  shared-jobs ← Intersection(target-jobs, base-case)
  StdDev ←  Skeletal-Deviation(shared-jobs, base-case)
  qual ← [len(base-case)^2 − (StdDev • len(base-case))] • len(shared-jobs) / len(base-case)
  Return(qual)
}
```

This "evenness" factor is a measure of the distribution of the target jobs in the base job sequence. The lengths of segments of the base sequence that do not contain target jobs are obtained and the standard deviation of these lengths is calculated. Large deviations are indicative of uneven skeletons and result in poor quality (low values) measures[1].

So the function `Skeletal-Deviation` measures the unevenness of the overlap in a skeletal solution.

```
function Skeletal-Deviation(shared-jobs, base-case){
  position-list ← Positions(shared-job, base-case)
  difference-list ← Positional differences between adj. shared-jobs
  mean-difference ← (∑ diff, diff∈difference-list) / |difference-list|
  deviation ← (∑ (mean-difference − diff)^2, diff∈difference-list) / |difference-list|
  Return(deviation)
}
```

For example, consider the target specification (T) containing 10 jobs {0,1,2,3,4,5,6,7,8,9}. Suppose cases C1 and C2 encode the following schedules, {36,3,2,6,8,12,15,20,4,22} and {4,20,2,23,6,31,7,15,9,17}. Both cases share 5 jobs with the target. However the skeletal deviation of C1 is 1.68 (with a mean difference of 1.75) and for C2 it is 0 (with a mean of 2); the target jobs are evenly distributed through C2 but concentrated in the first half of C1. The result is that the quality of C1's skeleton is 41.6 while C2's is 50. In other words C2 is preferred over C1.

4.2 Case Retrieval for the Building Block Approach

The building block retrieval algorithm uses the same base filtering process as the skeletal retrieval algorithm (although the intersection threshold is generally set to a lower value). The main difference between the procedures is that instead of using a quality metric, building block retrieval calls a function which breaks a case solution up into reusable building block sequences (schedule bits). These bits are collected and those large enough (≥ 3 jobs) are returned to be used later in composing the final target solution.

```
function Retrieve-Building-Blocks(target-jobs, case-base) {
  bits ← {}
  base-cases ← Base-Filter(target-jobs, case-base)
  For each base-case in base-cases {
    bits ← bits + Extract-Bits(target-jobs, base-case)
  }
  Return (bits)
}
```

The first step in the bit extraction procedure (shown below) is to mark the end (and start) positions of each reusable bit by noting the positions of any jobs in the base case that are not in the target. The base solution sequences between these positions are reusable as building

blocks because they contain only target jobs. If such a block contains 3 or more jobs then it is returned by the `Extract-Bits` function.

```
function Extract-Bits(target-jobs, base-case){
  bits ← {}
  unwanted-jobs ← Set-Difference(base-case, target-jobs)
  bit-ends ← Positions(unwanted-jobs, base-case)
  bit-start ← First(bit-ends)
  For each bit-end in Rest(bit-ends) {
    bits ← bits + Extract-Bit(base-case, bit-start, bit-end)
    bit-start ← bit-end
  }
  Return(Bits)
}
```

Figure 1 shows how reusable bits are extracted from C1 and C2 (the cases of the previous example) according to the jobs specified in the target sequence T. The unshaded jobs represent the edges of reusable bits and the shaded sections (which contain only target jobs) can be extracted for potential reuse. The interesting thing to note is that C1 is far more useful than C2 when the building block approach is used; this is in complete contrast to the skeletal approach where C2 was deemed to be more useful. In fact in this example C2 yields no useful bits (> 2 jobs in length) where as C1 yields a 4-job bit.

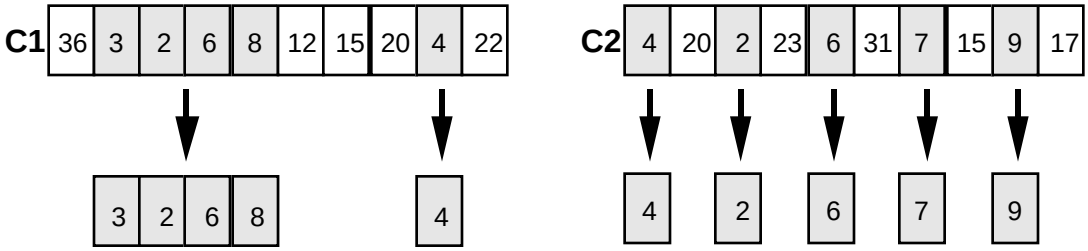


Figure 1. Building Block extraction.

The graph in Figure 2 summarises the results of building block retrieval, averaged over 100 test retrievals, on a case-base of 600 cases, each encoding a 100 job schedule. It turns out that the number of bits retrieved of a given size depends exponentially on that size. So while a typical retrieval may return nearly 200 3-job bits, it returns less that 50 4-job bits, about 10 5-job bits, and usually only 1 or 2 6-job bits; very rarely 7-job or 8-job bits may also be found.

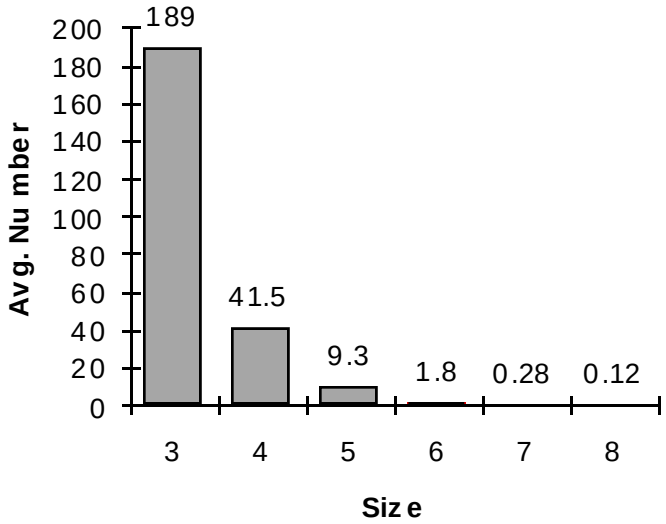


Figure 2. Average numbers of blocks retrieved for the Building Block solution

5 System Evaluation

Two issues are addressed in the evaluation process. We compare the solution quality and solution time of the two CBR techniques with the two alternatives mentioned already. We also evaluate how these techniques perform as the size of the target problem is increased. This second evaluation is particularly relevant because the time taken by the annealing technique increases more than linearly with the problem size.

5.1 Solution Quality & Time

In evaluating medium quality solution techniques to TSP problems it is worth considering a simple measure that will often improve solutions. Each pair of jobs can be considered in turn to see if reversing the section of schedule between these two points produces an improved solution. We call this gradient descent process 'Freezing' by analogy with what happens in Simulated Annealing. This measure does not produce any significant improvement in the SA solution; however it does improve the other three. Thus in evaluating solution time and quality we consider seven alternatives in total.

As mentioned already, the case-base in use in this evaluation contains 600 good quality schedules produced using SA. This size of case-base is chosen so that a case overlapping a target problem by 30% can expect to be retrieved. Figure 3 shows data drawn from running the seven solution techniques on 100 different target problems. The average times and average schedule lengths were calculated. The times show SA taking about 30 times longer than the CBR alternatives.* In turn, the Myopic solution time is negligible compared to these.

The solution quality is plotted as a percentage above the average solution obtained with SA. The CBR solutions are within 2% to 2.3% of the SA solution with the Building Block with Freezing solution improving to within 1% of SA. The Myopic solution is 5% above the best value and improves to 2.5% with freezing. So the BB+Freezing solution emerges as a source of good quality solutions in a fraction of the time taken by SA.

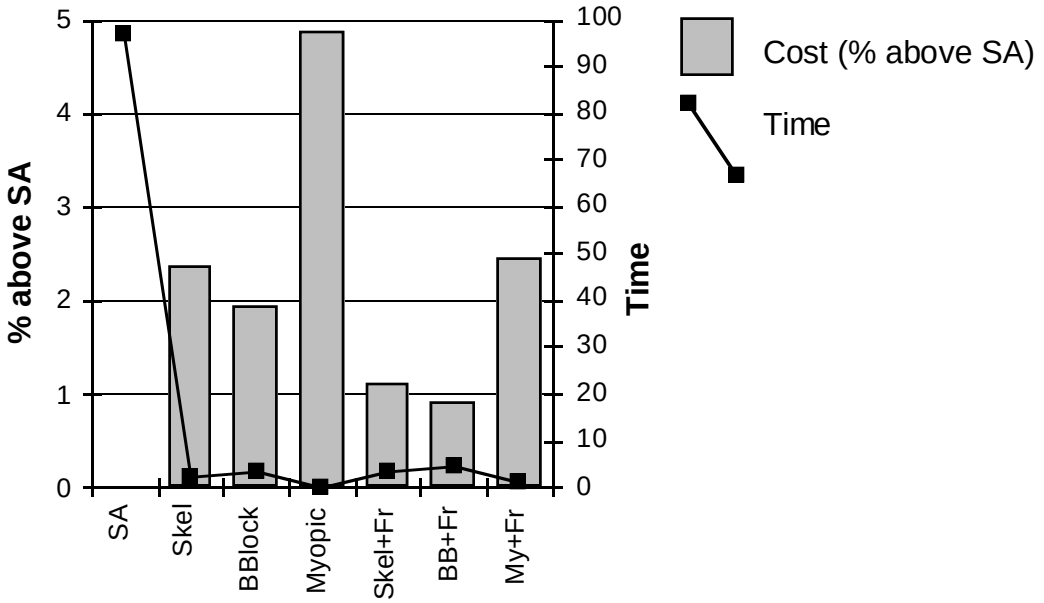


Figure 3. An evaluation of the solution time and solution quality of the different algorithms.

5.1 Variations in schedule size

It is clear from an examination of the workings of the CBR techniques that they can be used to produce solutions for problems of a size different to those in the case-base. In this part of

the evaluation we consider target problems varying in size from 40 to 140 in steps of 10; 20 problems are analysed at each step. It can be seen in Figure 4 that the skeletal solution quality remains more or less constant at around 2.5% above the SA values. The building block solutions appear to disimprove slightly with problem size.

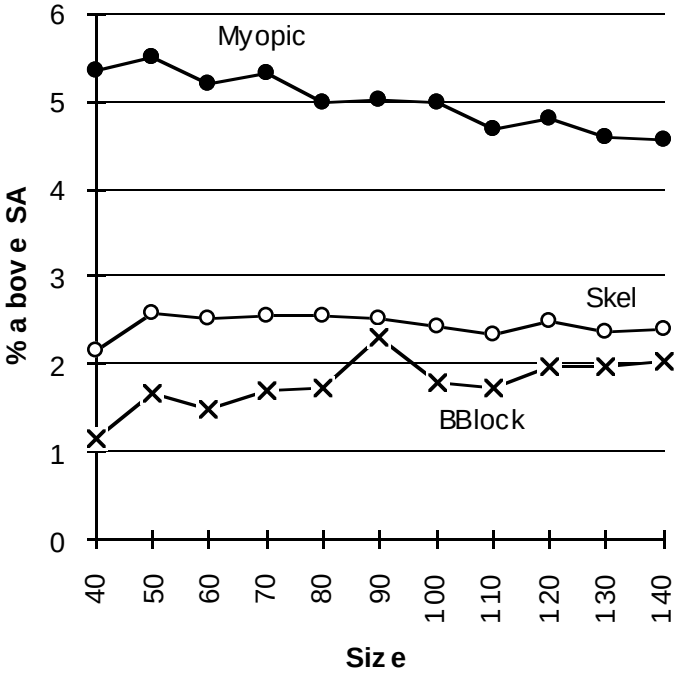
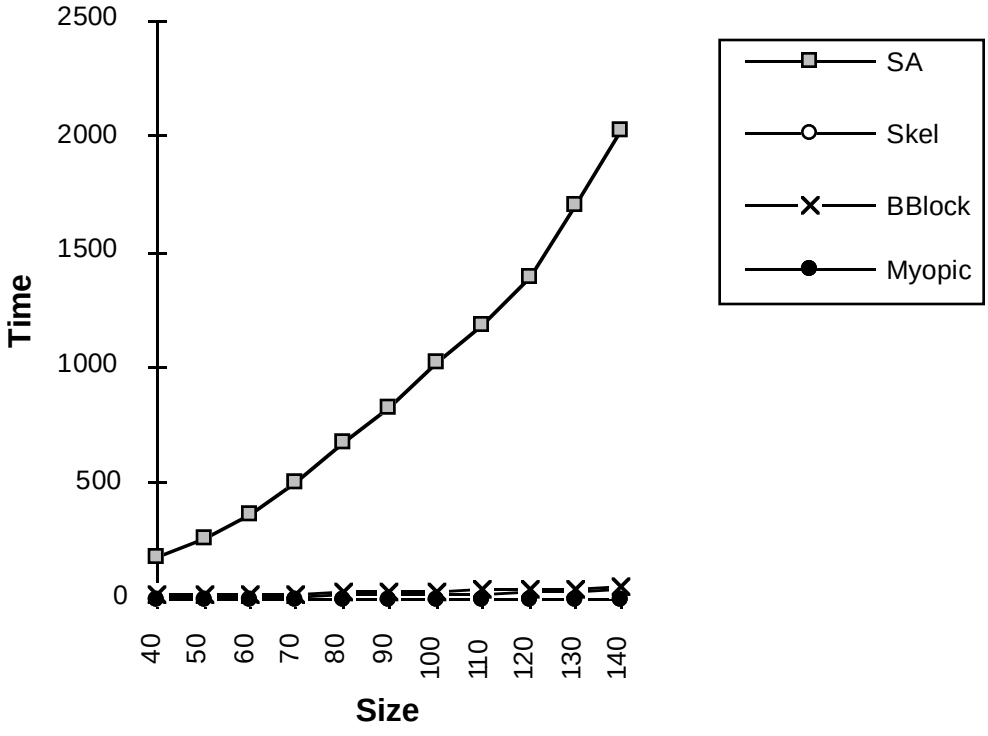


Figure 4. Variations in solution quality with target problem size.

Perhaps of more interest in this analysis of performance as problem size increases is the time performance of the techniques (see Figure 5). The SA time appears to increase more than linearly with problem size whereas the times for the CBR solutions are better behaved. At 40 jobs the SA takes 10 times as long as the BB solution; at 140 jobs it takes nearly 40 times as long.



6 Conclusions & Future Work

The evaluation described in this paper shows that for single machine scheduling problems with sequence dependent setup time CBR techniques can be used to produce good quality schedules. Known good schedules can be stored in a case-base and portions of these schedules can be reused to build new schedules in very quick time. We have evaluated two models of reuse; a skeletal technique whereby one case is reused, and a building block approach where several solutions are merged. We argue that this building block approach should be usable in other more complex scheduling problems.

There are two caveats however. It appears that schedule adaptation will be highly problem dependent. In new scheduling scenarios the success of these reuse-based techniques will depend on discovering good adaptation methods. Case retrieval time will probably increase linearly with case-base size. This will be an issue where large case-bases are required to give good coverage.

Our next step is to evaluate this building block technique on more complex scheduling problems.

References

Cunningham P., Smyth B., (1995) On the use of CBR in optimisation problems such as the TSP, Trinity College Dublin Computer Science Technical Report TCD-CS-95-15.

Cunningham P., Browne J., (1986) A LISP-based heuristic scheduler for automatic insertion in electronics assembly, *International Journal of Production Research*, Vol.24, No.6, pp1395-1408.

Goldberg D.E., (1989) *Genetic Algorithms in Search Optimization & Machine Learning*, Addison Wesley, Reading Massachusetts.

Kirkpatrick S., Gelatt C.D., Vecchi M.P., (1983) Optimization by Simulated Annealing, *Science*, Vol. 220, No. 4597, pp671-680.

Koton, P. (1989) SMARTplan: A Case-Based Resource Allocation and Scheduling System. *Proceedings of the Case-Based Reasoning Workshop*, pp 285-289, Florida, USA.

Muñoz H., Paulokat J., Wess S., (1994) Controlling Nonlinear Hierarchical Planning by Case Replay, in working papers of the *Second European Workshop on Case-Based Reasoning*, pp195-203, Chantilly, France.

Norback J., Love R., (1977) Geometric Approaches to Solving the Travelling Salesman Problem, *Management Science*, July 1977, pp1208-1223.

Smyth B., Cunningham P., (1992) Déjà Vu: A Hierarchical Case-Based Reasoning System for Software Design, in Proceedings of *European Conference on Artificial Intelligence*, ed. Bernd Neumann, John Wiley, pp587-589.

Sycara, K. & Miyashita, K. (1994) Case-Based Acquisition of User Preferences for Solution Improvement in Ill-Structured Domains. *Proceedings of the 12th National Conference on Artificial Intelligence*, pp. 44-49. Seattle, USA.